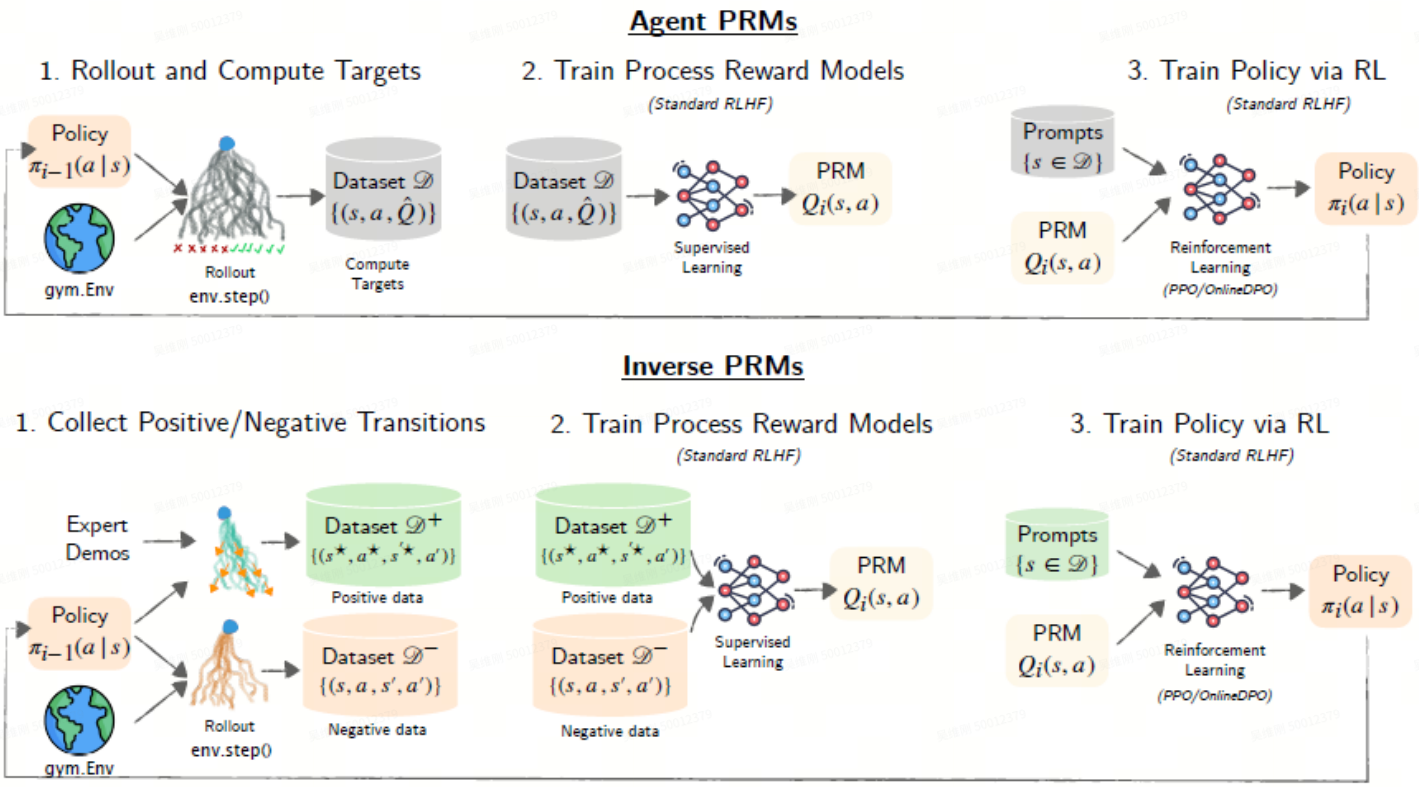
面向LLM Agent的过程奖励模型 (PRM)

Agent PRM

显式PRM

Process reward models for llm agents: Practical framework and directions [PRM4A]



1. AgentPRM

   Stage 1：采样和计算当前Q值。根据当前的策略去采样的到数据：

$$\{(s_0, a_0, r_0, \ldots, s_{T-1}, a_{T-1}, r_{T-1})\}.$$

$$\hat{Q}(s,a) = \frac{1}{|\mathcal{G}(s,a)|} \sum_{(s_t,a_t)\in\mathcal{D}(s,a)} \sum_{k=t}^{T-1} \gamma^{k-t} r_k$$

   Stage 2：训练PRM。

$$\mathcal{L}(Q_\phi) = -\mathbb{E}_{(s,a,\hat{Q})\sim\mathcal{D}} \left[\hat{Q} \log Q_\phi(s,a) + (1-\hat{Q}) \log(1 - Q_\phi(s,a))\right]$$

Stage 3：训练RL。（注意KL不是reference策略，而是上一步i-1的策略，因为PRM是根据上一步策略采样的数据训练得到的）

$$\pi_i = \arg\max_{\pi_\theta} \mathbb{E}_{s\sim\mathcal{D},a\sim\pi_\theta(a|s)} \left[Q_\phi(s,a)\right] - \beta\mathbb{D}_{\mathrm{KL}} \left[\pi_\theta(a|s)||\pi^{i-1}(a|s)\right]$$

2. InversePRM：假设是有专家数据的情况下

Stage 1：构建正负样本。专家数据为正样本，当前策略采样的数据为负样本。

Stage 2：训练PRM。还是借用GAN的思想。

$$r(s,a) = Q^\pi(s,a) - \gamma\mathbb{E}_{a'\sim\pi}Q^\pi(s',a').$$

Train PRM Q_i by minimizing the classification loss:

$$\begin{aligned}\mathcal{L}(\phi) = &-\mathbb{E}_{(s^*,a^*,s'^*,a')\sim\mathcal{D}^+}\left[\log\sigma(Q_\phi(s^*,a^*) - \gamma Q_\phi(s'^*,a'))\right] \\ &+\mathbb{E}_{(s,a,s',a')\sim\mathcal{D}^-}\left[\log(1-\sigma(Q_\phi(s,a) - \gamma Q_\phi(s',a')))\right]\end{aligned}$$

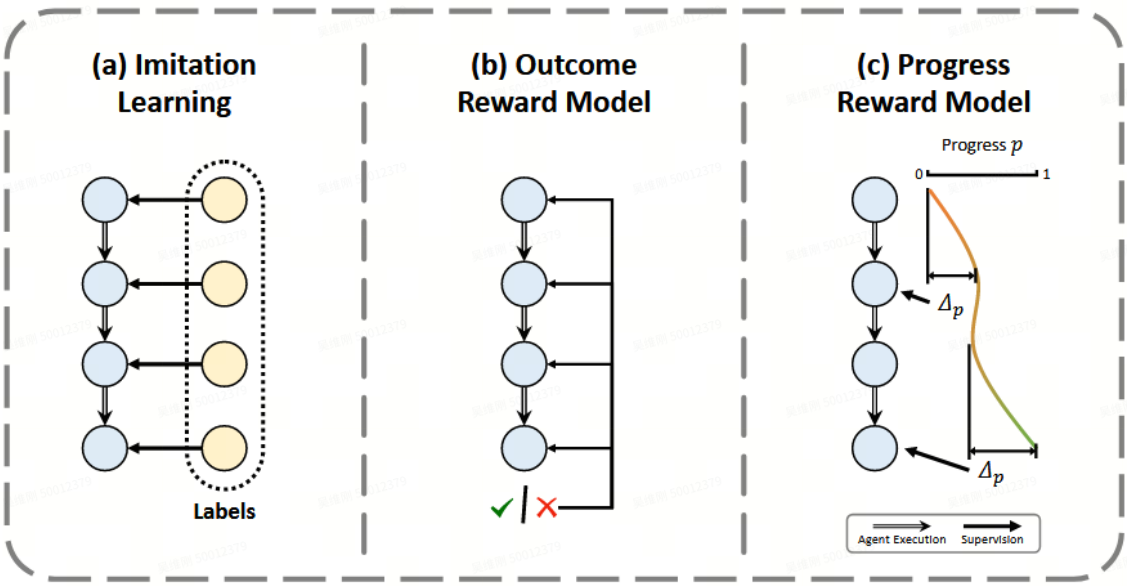
Q判别器用尽可能区分出正负样本的数据来源；

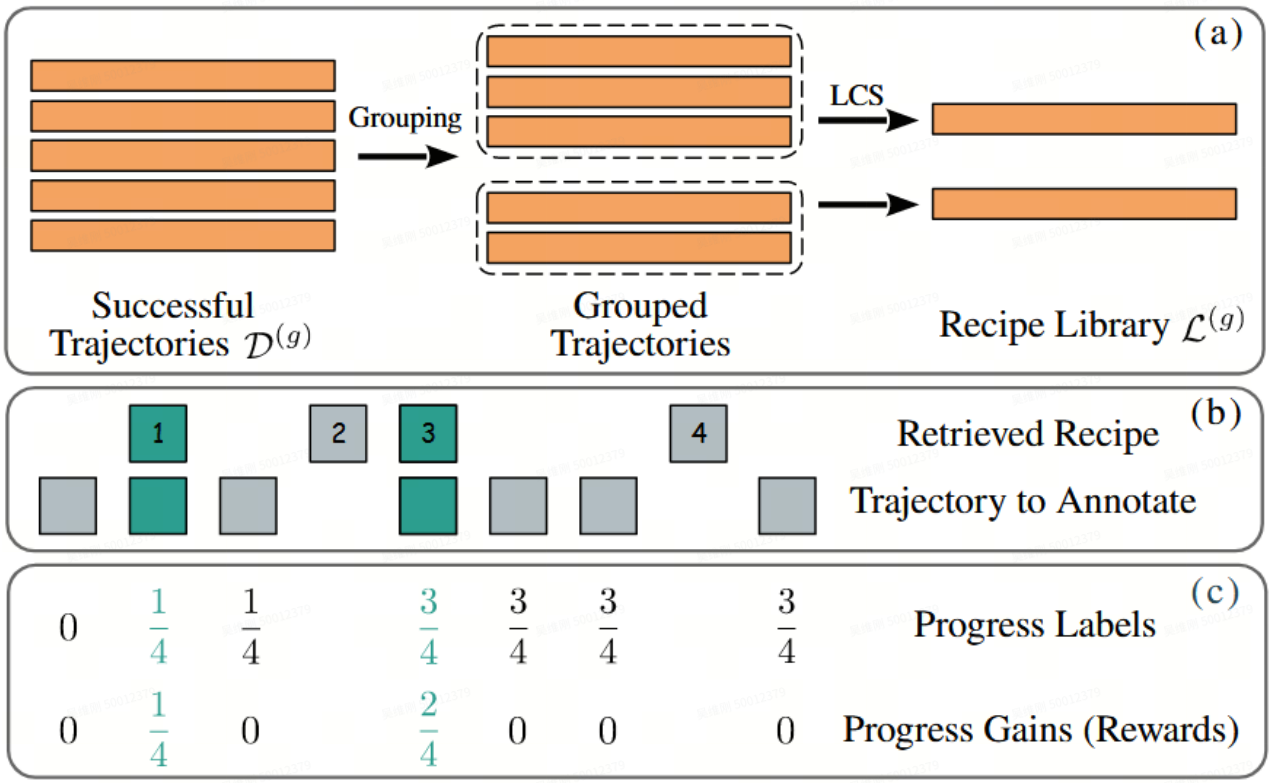
π_θ 以 $Q_\phi(s,a)$ 作为单步奖励来更新策略，奖励越大代表判别器越认为其是专家策略。

Stage 3：训练RL。（注意KL不是reference策略，而是上一步i-1的策略，因为负样本是根据上一步策略采样）

$$\pi_i = \arg\max_{\pi_\theta} \mathbb{E}_{s\sim\mathcal{D},a\sim\pi_\theta(a|s)} \left[Q_\phi(s,a)\right] - \beta\mathbb{D}_{\mathrm{KL}} \left[\pi_\theta(a|s)||\pi^{i-1}(a|s)\right]$$

ProgRM: Build Better GUI Agents with Progress Rewards





提出了一个名为“进度奖励模型”的方法。这个模型在agent执行任务的每一步都预测出当前任务的完成进度，以计算每一步带来的“进度增益”，为强化学习提供密集的奖励信号。

1. 进度奖励定义

进度定义为任务完成度的百分比。在任务g的执行过程中，agent在状态s_t的进度被表示为：

$$p_t = \mathrm{Prog}(s_t; g).$$

Prog为要训练的进度奖励模型。基于这个进度估计，agent在时刻t获得的进度奖励被定义为：

$$r_t^{(p)} = \mathrm{Prog}(s_t; g) - \mathrm{Prog}(s_{t-k}; g),$$

当前进度和k步前的进度的差值，论文中k被设置为1。

2. 过程标签的自动生成

他们设计了一个无需人工标注、自动生成标注标签的方法。

- 构建LCS库。LCS是最长公共子序列。他们假设成功执行任务的轨迹拥有一个共同的核心行为模式，LCS就是完成该任务的关键步骤序列。因此，需要找出所有任务的关键步骤序列（LCS）。首先搜集大量成功的执行轨迹，然后把这些轨迹通过相似度计算分成若干组。在组内的轨迹之间计算LCS，形成一个执行配方（execution recipe）库。
- 关键步骤发现。对于任意一条需要标注的轨迹（无论成功与否），用配方库中的所有配方去匹配它，匹配度最高的那个配方被选中，轨迹中与配方成功匹配的步骤被识别为“关键步骤”。

- 进度标签分配。关键步骤的进度标签根据其在配方中的位置进行分配，例如，一个关键步骤是配方（长度为|L|）中的第κ个元素，那么它的进度标签就是 κ/|L|。 而那些非关键步骤，则继承其前一个最近的关键步骤的进度标签。 这样，轨迹中的每一步都有了一个进度标签 `p_t*` 。
- 进度奖励模型的训练。有了标注数据{（g_i, s_i, p_i*）}，通过交叉熵损失训练奖励模型。

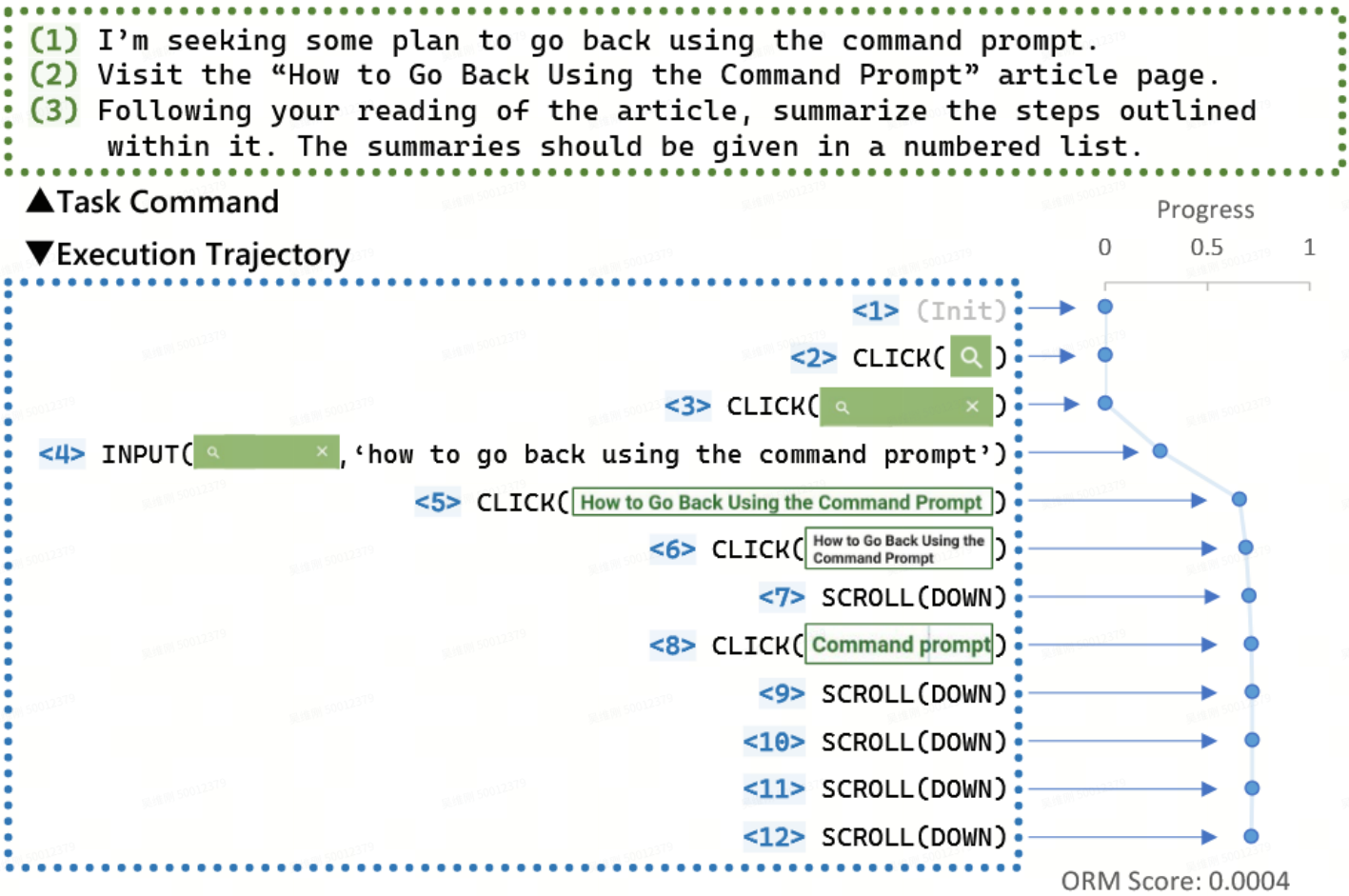
$$\hat{p}_i = \text{Prog}([g_i, \hat{s}_i]; \theta),$$
$$\mathcal{L}(\theta) = \mathbb{E}_{i \sim \mathcal{D}^{(p)}} \left[-p_i^* \log \hat{p}_i - (1 - p_i^*) \log(1 - \hat{p}_i) \right].$$

3. 在线强化学习

采用REINFORCE++算法来训练agent。

问题：

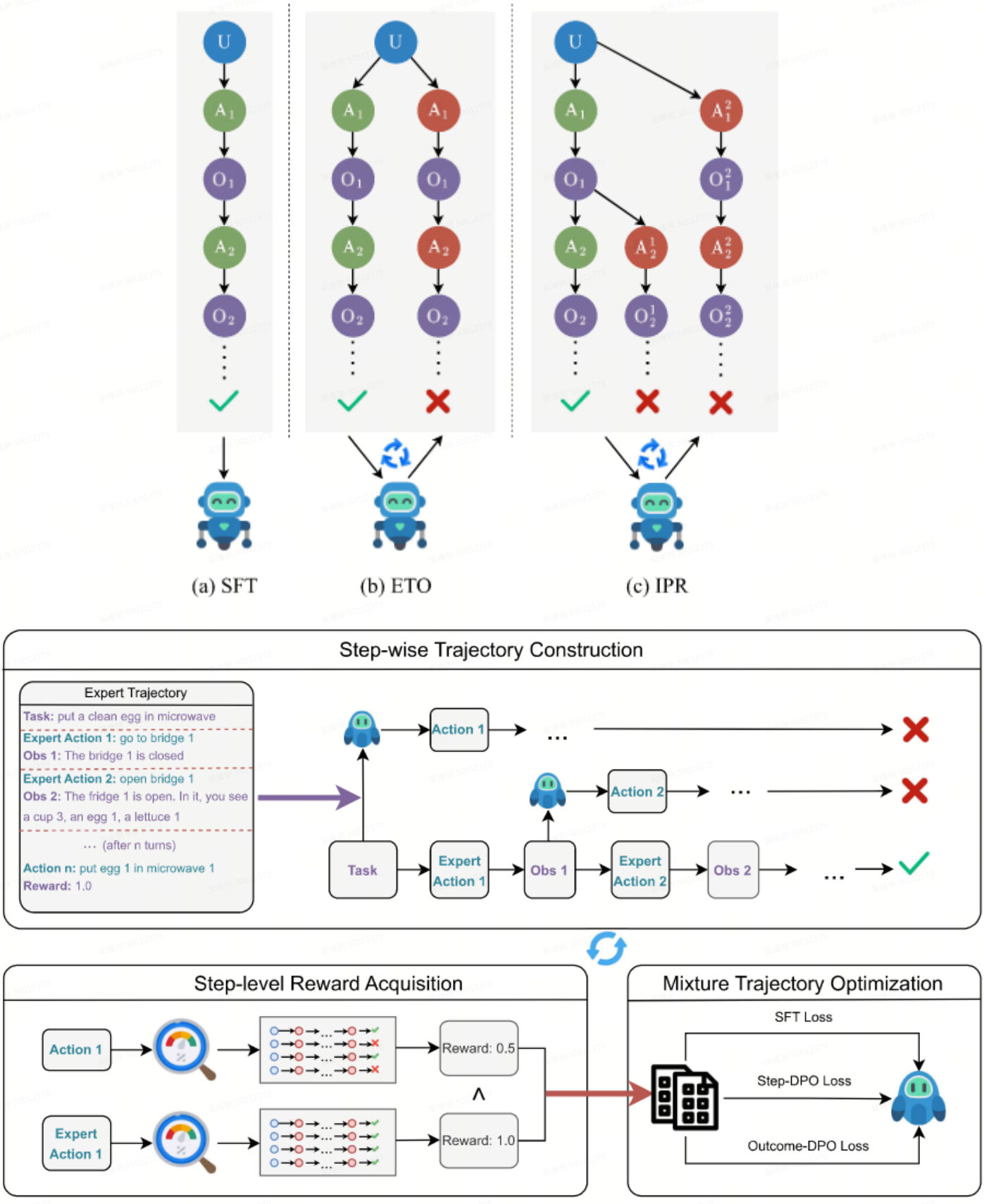
他们过程标签生成的方法对数据集有限制，他们的任务的执行动作，限制在几个原子动作之内，可以用他们手动设计的相似度规则来匹配。但是让动作扩大到复合动作、自然语言、可执行代码，就很难设计这种规则来匹配。



隐式PRM（细粒度奖励）

Watch Every Step! LLM Agent Learning via Iterative Step-Level Process Refinement

EMNLP main



思路：

在智能体沿循一条专家轨迹执行任务时，在某一步 t，故意让智能体尝试一个不同于专家的“探索性”动作。然后，通过蒙特卡洛（Monte Carlo）模拟，分别评估从“专家动作”和“探索性动作”出发，继续执行任务直到结束，哪一个能带来更好的最终结果。

这个对比结果直接构成了一个学习信号，告诉策略模型：“在这个节点，专家动作比你探索的那个动作要好（或差）”。通过在任务的各个步骤上重复这个过程，模型就能学会每一步的正确决策，从而实现精细的信用分配。

方法：

- 步级奖励获取（Step-level Reward Acquisition）

给定一个专家轨迹，在轨迹的第t步，让当前的策略模型在同样的状态下，生成一个或多个备选动作。对于专家动作和备选动作，都从当前状态出发执行直到任务结束，记录获得的奖励。

- 混合轨迹优化（Mixture Trajectory Optimization）

IPR不使用这个计算出的奖励值来训练一个单独的PRM网络。相反，它将上述的对比结果直接转化为偏好数据，用于策略模型的优化。

他们的Loss函数分为三部分

Outcome-DPO Loss, L_{o-DPO} ,

$$\mathcal{L}_{\text{o-DPO}} = -\mathbb{E}_{(u,e_n^w,e_m^l)\sim\mathcal{D}_t}\Bigg[\log\sigma(\beta\log\frac{\pi_\theta(e_n^w|u)}{\pi_{ref}(e_n^w|u)} -\beta\log\frac{\pi_\theta(e_m^l|u)}{\pi_{ref}(e_m^l|u)})\Bigg],$$

Step-DPO Loss, L_{s-DPO}

$$\mathcal{L}_{\text{s-DPO}} = -\mathbb{E}_{(e_{t-1},e_{t:n}^w,e_{t:m}^l)\sim\mathcal{D}_s}\Bigg[\log\sigma(\beta\log\frac{\pi_\theta(e_{t:n}^w|e_{t-1})}{\pi_{ref}(e_{t:n}^w|e_{t-1})} -\beta\log\frac{\pi_\theta(e_{t:m}^l|e_{t-1})}{\pi_{ref}(e_{t:m}^l|e_{t-1})})\Bigg],$$

SFT Loss, L_{SFT}

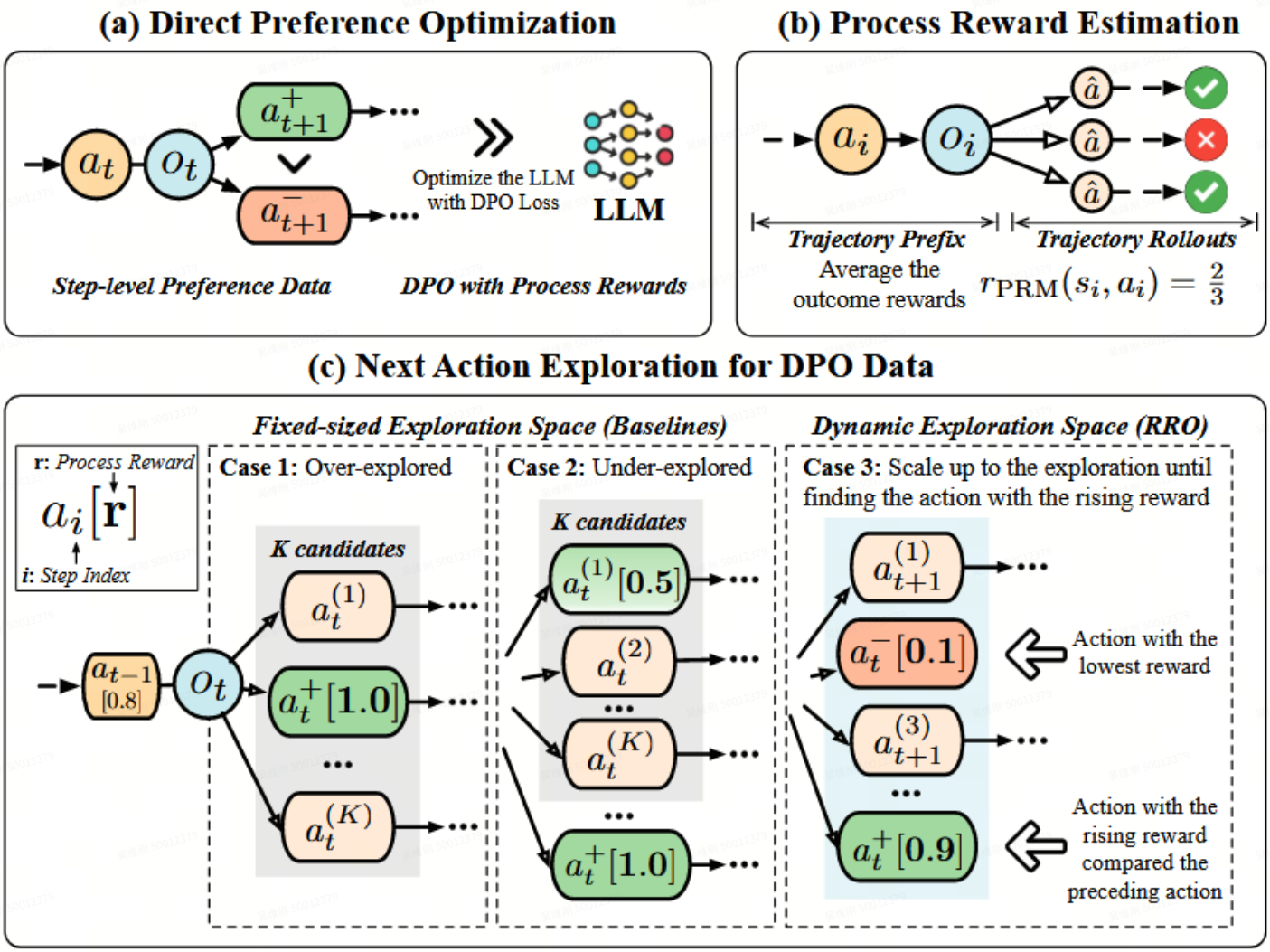
$$\mathcal{L}_{\text{SFT}} = -\mathbb{E}_{(u,e_n^w,e_m^l)\sim\mathcal{D}_t}\Bigg[\log\pi_\theta(e_n^w|u)\Bigg],$$

最后的损失函数是这三部分相加：

$$\mathcal{L} = \mathcal{L}_{\text{o-DPO}} + \mathcal{L}_{\text{s-DPO}} + \mathcal{L}_{\text{SFT}}$$

Paradigm	Models	WebShop	InterCodeSQL	ALFWorld		Average
				Seen	Unseen	
Prompt-based	GPT-4 (Achiam et al., 2023)	63.2	38.5	42.9	38.1	45.7
	GPT-3.5-Turbo (Ouyang et al., 2022)	62.4	37.8	7.9	10.5	29.7
	Llama-2-7B (Touvron et al., 2023)	17.9	4.0	0.0	0.0	5.5
Outcome Refinement	Llama-2-7B + SFT (Chen et al., 2023)	60.2	54.9	60.0	67.2	60.6
	Llama-2-7B + PPO (Schulman et al., 2017)	64.2	52.4	22.1	29.1	42.0
	Llama-2-7B + RFT (Yuan et al., 2023)	63.6	56.3	62.9	66.4	62.3
	Llama-2-7B + ETO (Song et al., 2024)	67.4	57.2	68.6	72.4	66.4
Process Refinement	Llama-2-7B + Step-PPO	64.0	60.2	65.7	69.4	64.8
	Llama-2-7B + IPR (ours)	**71.3**	**61.3**	**70.3**	**74.7**	**69.4**

RRO: LLM Agent Optimization Through Rising Reward Trajectories



RRO（Reward Rising Optimization）提出了一种动态的、智能化的数据采样策略。其核心思想是，在为智能体构建训练数据时，我们不必在每一步都详尽地探索所有可能的下一步动作来寻找那个“绝对最优”的动作。相反，我们的目标应该是找到一个**比当前步骤更好的动作**。

该方法设定了一个“奖励提升”（Reward Rising）的准则：在探索下一步候选动作时，一旦发现某个候选动作的过程奖励**不低于**上一步已执行动作的过程奖励，就停止探索。然后，从已探索的这部分候选动作中，选出奖励最高的作为“更优”样本，奖励最低的作为“更差”样本，构成一个偏好对（preference pair），用于后续的智能体优化。

1. 过程奖励定义

$$r_{\text{PRM}}(s_t, a_t) = \frac{1}{m}\sum_{j}^{m} r_{\text{ORM}}(u, e_{1:t} \oplus \hat{e}_{t+1:n}^{(j)})$$

`r_PRM(s_t, a_t)` 的值，就是从 `(s_t, a_t)` 出发，未来能够成功完成任务的**平均概率**或**期望成功率**。

2. 奖励提升采样

假设智能体刚执行完动作 `a_{t-1}`，其过程奖励已经计算出来，为 `r_{PRM, t-1}`。现在需要为下一步 `t` 生成训练数据：

- **启动探索：** 使用当前策略 `π_θ` 生成第一个候选动作 `â_t^(1)`。
- **昂贵评估：** 对 `â_t^(1)` 执行上述的MCTS过程，计算出其过程奖励 `r_{PRM, t}^(1)`。
- **检查停止条件：** 比较 `r_{PRM, t}^(1)` 和 `r_{PRM, t-1}`。
  - **如果** `r_{PRM, t}^(1) < r_{PRM, t-1}`：说明这个候选动作甚至不如上一步，我们可能需要更好的选择。于是继续采样第二个候选动作 `â_t^(2)`，计算其奖励，并重复此检查。
  - **如果** `r_{PRM, t}^(τ) >= r_{PRM, t-1}`：一旦找到第 `τ` 个候选动作，其奖励终于“提升”或持平了，**立刻停止探索过程**。
- **构建偏好对：** 在所有已经采样的 `τ` 个候选动作 `{(â_t^(i), r_{PRM, t}^(i))}` 中：
  - 找到奖励最高的动作作为**更优动作 (winning action)** `y_w`。
  - 找到奖励最低的动作作为**更差动作 (losing action)** `y_l`。
  - 将 `(y_w, y_l)` 作为一个偏好数据点存储起来。

3. 用DPO来微调LLM

$$\mathcal{L}_{\text{DPO}}(\pi_\theta) = -\mathbb{E}_{(x,y_w,y_l)}\left[\log\left(\sigma\left(\beta\log\frac{\pi_\theta(y_w)}{\pi_{\text{ref}}(y_w)} - \beta\log\frac{\pi_\theta(y_l)}{\pi_{\text{ref}}(y_l)}\right)\right)\right]$$

Algorithm 1: Reward Rising Optimization (RRO)

Data: u is the task instruction; π_θ is the supervised fine-tuned policy model; $e_{1:t}$ is the trajectory prefix from step 1 to t; m is the number of samples in Monte-Carlo Tree Search.

Function `ProcessReward`$(u, e_{1:t} = [u, a_1, o_1, ..., a_t, o_t], \pi_\theta, m)$**:**
- $MC \leftarrow [\,]$
- **for** $i = 1$ *to* m **do**
  - $\hat{e}_{t+1:n} \sim \pi_\theta(\cdot|e_{1:t})$ ▷ Sample the rollout $\hat{e}_{t+1:n}$ from the policy model π_θ
  - $e \leftarrow e_{1:t} \oplus \hat{e}_{t+1:n}$ ▷ Concatenate the trajectory prefix and rollout to have the full trajectory e.
  - MC.append$(r_{ORM}(u, e))$ ▷ Evaluate the outcome reward based on the full trajectory.
- **end**
- $r_{PRM} \leftarrow \frac{1}{m} \sum_{r_{\text{ORM}} \in MC} r_{\text{ORM}}$ ▷ Estimate the process reward with the average outcome rewards.

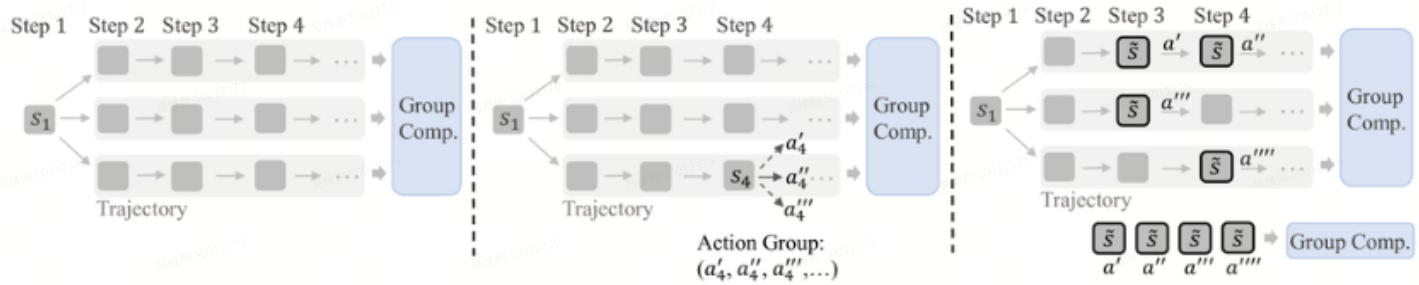
return r_{PRM}

Function `RRO`(u, π_θ, t)**:**
- $e_{1:t-1} = [u, a_1, o_1, ..., a_{t-1}, o_{t-1}] \sim \pi_\theta(u)$ ▷ Sample $t-1$ steps as the trajectory prefix.
- $r_{PRM,t-1} \leftarrow$ ProcessReward$(u, e_{1:t-1}, \pi_\theta, m)$ ▷ Estimate the process reward via Monte-Carlo Sampling.
- $Cand \leftarrow [\,]$
- **repeat**
  - $\hat{a}_t \sim \pi_\theta(\cdot|e_{1:t-1})$ ▷ Explore the candidates for the next action.
  - $e_{1:t} \leftarrow e_{1:t} \oplus \hat{a}_t$
  - $r_{PRM,t} \leftarrow$ ProcessReward$(u, e_{1:t}, \pi_\theta, m)$ ▷ Estimate the process reward for the next action candidate.
  - $Cand$.append$((\hat{a}_t, r_{PRM,t}))$
- **until** $r_{PRM,t} \geq r_{PRM,t-1}$ ▷ Stop the exploration until showing a rising reward compared to a_{t-1}.
- $a_t^+ \leftarrow \arg\max_{r_{\text{PPM},t}} \{\hat{a}_t | (\hat{a}_t, r_{PRM,t}) \in Cand\}$
- $a_t^- \leftarrow \arg\min_{r_{\text{PPM},t}} \{\hat{a}_t | (\hat{a}_t, r_{PRM,t}) \in Cand\}$
- $\pi_\theta \leftarrow \text{DPO}(\pi_\theta, (a_t^+, a_t^-))$ ▷ Update the policy model with the preference pair from the sampling.

Base Model	Method	Agent Planning			
		WebShop		InterCode-SQL	
		Reward ↑	# Sample ↓	Reward ↑	# Sample ↓
Gemma-2_{2B}	*Without Post-training*				
	+ Few Shot	12.62	0	3.86	0
	Outcome Supervision				
	+ SFT (Chen et al., 2023a)	48.94	0	45.33	0
	+ ETO (Song et al., 2024)	52.34	1	47.13	1
	Process Supervision				
	+ IPR (Xiong et al., 2024)	<u>61.39</u>	4	52.39	3
	+ Fixed-sized Exploration	61.20	5	<u>54.68</u>	5
	+ Reward Rising Optimization (RRO)	**62.91** (+1.52)	1.86	**55.08** (+0.40)	1.64

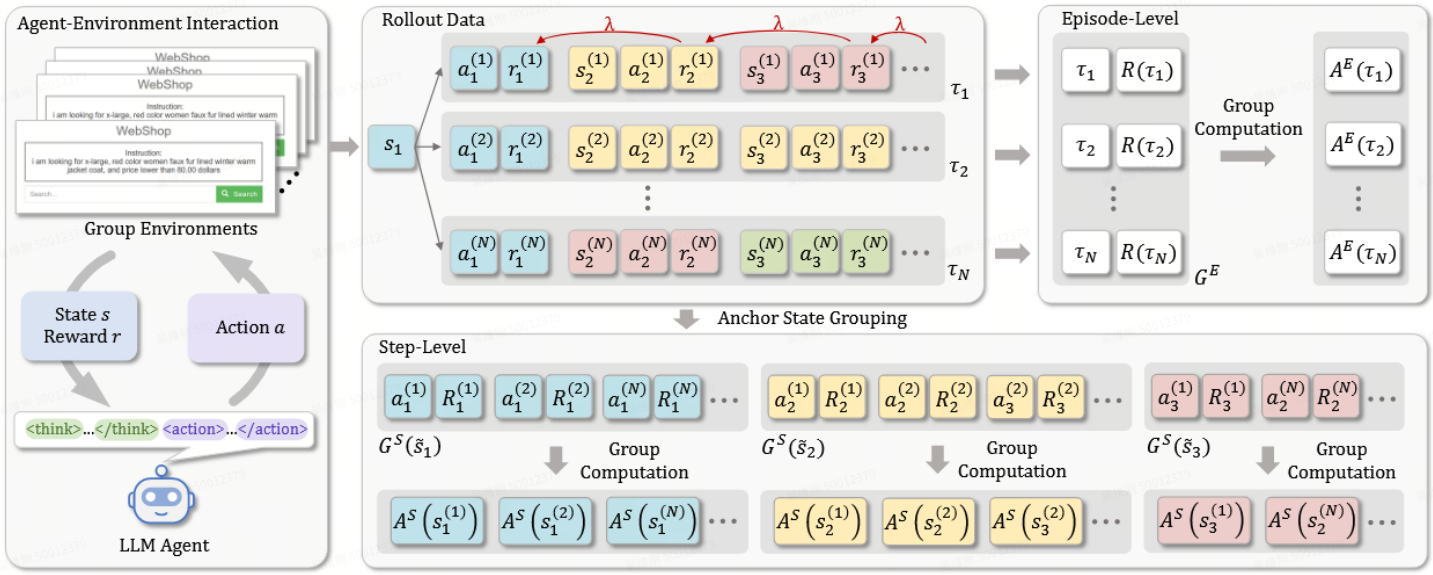
Group-in-Group Policy Optimization for LLM Agent Training

现有的GRPO方法在LLM中训练很高效，但是在agent的长程规划任务中，将整个轨迹视为一个整体进行打分，无法评价每一步的好坏，进行细粒度的信用分配，限制了他们的效果。而其他细粒度的方法需要额外的模型辅助计算Value，效率较低。



论文提出了GiGPO的方法，其核心是一个两级分组的优势估计结构：

1. episode相对优势，用来捕捉每个轨迹的整体有效性，提供全局的训练信号。
2. step相对优势，关注哪些动作在同一状态下优于其他动作，赋予细粒度的信号。



episode相对优势：计算方式与GRPO相同，以整个轨迹为单位，最终奖励为0或1

$$G^E = \left\{ (\tau_1, R(\tau_1)), (\tau_2, R(\tau_2)), \ldots, (\tau_N, R(\tau_N)) \right\}.$$

用组群相对优势的方法估计每一个轨迹的优势

$$A^E(\tau_i) = \frac{R(\tau_i) - \text{mean}\left(\{R(\tau_j)\}_{j=1}^N\right)}{F_{\text{norm}}\left(\{R(\tau_j)\}_{j=1}^N\right)}.$$

step相对优势：在收集到一批轨迹后，回顾在所有轨迹中出现的相同的环境状态，这些相同的状态被称为”锚点状态”，在这些状态下采取的不同的动作以及产生的未来回报作为一个“动作组”。

$$G^S(\tilde{\mathbf{s}}) = \left\{ \left(\mathbf{a}_t^{(i)}, R_t^{(i)} \right) \mid \mathbf{s}_t^{(i)} = \tilde{\mathbf{s}},\ 1 \le i \le N,\ 1 \le t \le T \right\}.$$

$$R_t^{(i)} = \sum_{k=t}^{T} \gamma^{k-t} r_k^{(i)}.$$

在这个动作组内部，计算出每个动作的微观相对优势

$$A^S(a_t^{(i)}) = \frac{R_t^{(i)} - \text{mean}\left(\left\{R_t^{(j)} \mid (a_t^{(j)}, R_t^{(j)}) \in G^S(\tilde{s})\right\}\right)}{F_{\text{norm}}\left(\left\{R_t^{(j)} \mid (a_t^{(j)}, R_t^{(j)}) \in G^S(\tilde{s})\right\}\right)}.$$

最终优势计算：用于指导模型更新的优势信号是这两个优势的加权和

$$A(a_t^{(i)}) = A^E(\tau_i) + \omega \cdot A^S(a_t^{(i)}),$$

ω 是平衡权重的超参数

$$\mathcal{J}_{\text{GiGPO}}(\theta) = \mathbb{E}_{\substack{x\sim p(X) \\ \{\tau_i\}_{i=1}^N \sim \pi_{\theta_{\text{old}}}}} \left[\frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T} \min\left(\rho_\theta(a_t^{(i)})A(a_t^{(i)}),\ \text{clip}\left(\rho_\theta(a_t^{(i)}), 1\pm\epsilon\right)A(a_t^{(i)})\right)\right] - \beta\mathbb{D}_{\text{KL}}\left(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\text{ref}}(\cdot \mid x)\right). \tag{9}$$

$$\rho_\theta(a_t^{(i)}) = \frac{\pi_\theta(a_t^{(i)} \mid s_t^{(i)}, x)}{\pi_{\theta_{\text{old}}}(a_t^{(i)} \mid s_t^{(i)}, x)}$$

优势：细粒度评估，无需critic model，无需额外rollout

Agentic Reinforced Policy Optimization

背景

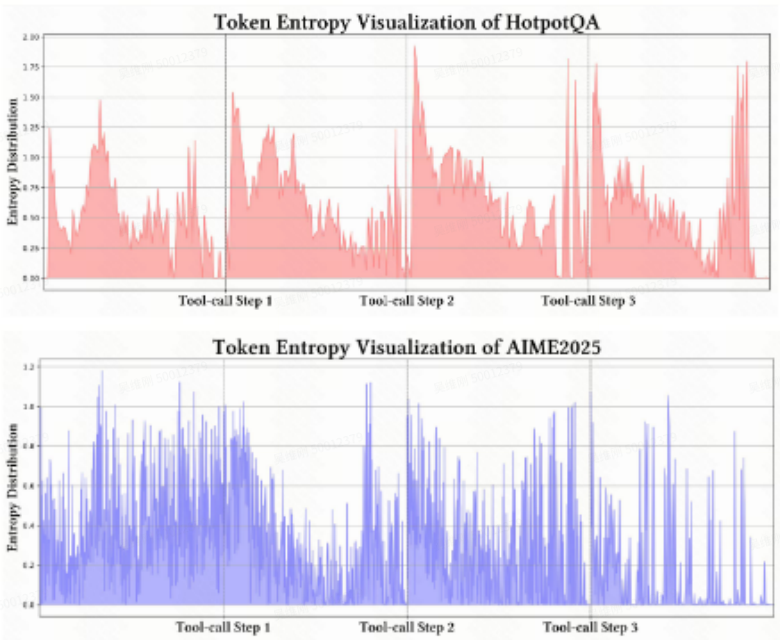
1. **从 RLVR 到 Agentic RL 的演进**：
   - **带可验证奖励的强化学习 (RLVR)** 在提升LLM单轮推理能力（如数学、编程）方面取得了巨大成功，催生了像GPT-4o、DeepSeek-R1等强大的模型。
   - 然而，现实世界的复杂任务（如深度调研、问题分析）需要LLM不仅仅是单次作答，而是要像一个智能体（Agent）一样，与外部环境（工具）进行**多轮、动态的交互**。
   - **智能体强化学习 (Agentic RL)** 应运而生，它将LLM的训练范式从静态的“解题”转变为动态的“智能体-环境”交互推理，这是AI发展的下一个重要阶段。
2. **现有 Agentic RL 算法的瓶颈**：
   - 目前主流的RL算法，如GRPO、DAPO等，大多采用**轨迹级 (Trajectory-level)** 的优化方式。
   - 这意味着算法会完整地采样一条从头到尾的工具使用轨迹（例如：思考 -> 搜索 -> 思考 -> 编程 -> 最终答案），然后根据最终结果的好坏给予一个整体的奖励信号。
   - 这种方法的弊端在于“**粗粒度**”：它忽略了在交互过程中每一个关键步骤的精细行为，无法有效探索在某个特定步骤下可能存在的更优选择。

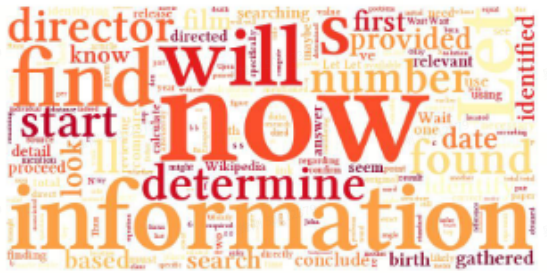
动机

作者观察到一个关键现象：当LLM Agent**调用一个外部工具（如搜索引擎）**并获得返回结果后，它接下来生成的token**熵（entropy）会显著飙升**。传统的RL算法（如GRPO）会均匀地对整个轨迹进行采

样，从而忽略了这些“高不确定性”的关键决策点。这篇文章则认为，这些**高熵时刻正是模型最需要学习和探索的地方**。

- **熵增的含义**：
  - **高不确定性**：高熵代表模型对于下一步要生成什么内容感到非常“迷茫”和“不确定”。工具返回的新信息打破了模型原有的思维链，使其进入一个充满可能性的决策岔路口。
  - **探索的关键节点**：这个高不确定性的时刻，正是模型最需要探索（Exploration）不同后续路径的关键节点。传统的轨迹级RL算法会随机选择一条路走到底，从而错过了在这些关键节点进行“分支探索”的绝佳机会。

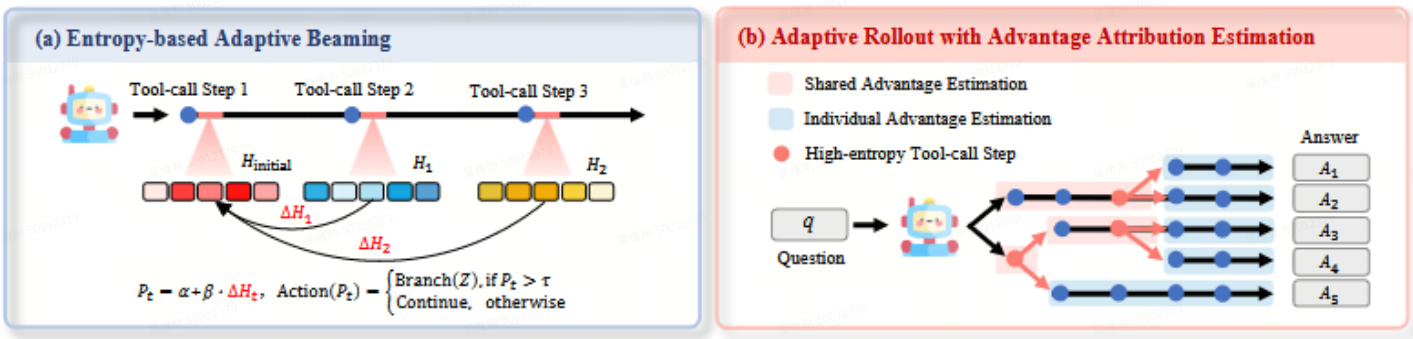




(a) Frequent tokens with the highest average entropy

(b) Frequent tokens with the lowest average entropy

方法



ARPO提出了一种**基于熵的自适应采样（Entropy-based Adaptive Rollout）**机制。它不再是简单地采样M条完整的轨迹，而是采用一种混合策略：

1. **全局采样：** 首先，像往常一样采样一小部分（例如N条）完整的轨迹，以保证探索的广度。

2. **分支采样：** 在采样过程中，ARPO会**实时监控**模型生成token的熵。一旦在某个工具调用之后，模型输出的token熵**超过一个预设的阈值**，ARPO就会认为这是一个关键的“分岔路口”。此时，它会利用剩余的采样预算M-N条，从这个“分岔点”开始，**分支（branch out）**出多条不同的后续路径（partial rollouts）进行深度探索。

最后，它通过一种巧妙的**优势归因估计（Advantage Attribution Estimation）**，利用GRPO算法的内在机制，来为这些共享了共同前缀但后续路径不同的轨迹，分配合理的奖励信号。

1. **基于熵的自适应采样：**

   在RL的Rollout阶段，

   - 初始化。设定全局采样预算为M条，其中N条用于全局采样，M-N条用于分支采样。计算一个初始的token熵

   $H_{initial}$

   - 熵监控。在生成轨迹时，算法持续计算接下来的生成的token的熵H_t相对于初始状态的归一化变化。

$$\Delta H_t = \text{Normalize}(H_t - H_{\text{initial}})$$

   - **分支决策：** 算法会计算熵的变化量 `ΔH_t` （公式4），并根据这个变化量来决定是否进行分支。一个简单的决策规则是：
     - `P_t = α + β * ΔH_t`，其中 `P_t` 是分支概率， `α` 和 `β` 是超参数。
     - 如果 `P_t` 超过阈值 `τ`，则触发**分支**，从当前点开始，生成 `Z` 条不同的后续路径。
     - 如果不超过，则继续沿着当前路径生成。

$$P_t = \alpha + \beta \cdot \Delta H_t, \quad \text{Action}(P_t) = \begin{cases} \text{Branch}(Z), & \text{if } P_t > \tau \\ \text{Continue}, & \text{otherwise} \end{cases}$$

2. **优势归因估计**

在进行了分支采样后，如何让模型从这些既有共享前缀又有不同分支的轨迹中有效学习，是下一个关键问题。

Hard Advantage Estimation

对于所有轨迹，用归一化奖励计算单个token的优势

$$\hat{A}_{i,t} = \frac{r_i - \text{mean}(\{R_i\}_{i=1}^{G})}{\text{std}(\{R_i\}_{i=1}^{G})}.$$

对于共享的token，在包含共享段的d个轨迹中分配平均优势

$$\hat{A}_{i,t}^{\text{shared}} = \frac{1}{d} \sum_{i=1}^{d} \hat{A}_{i,t}.$$

这种方法非常直观，但实现起来需要显式地追踪和区分共享与非共享部分。一种隐式的方法是软优势估计。

Soft Advantage Estimation

软优势估计的核心是，

- 对于在分支点**之前**的、所有轨迹都**共享**的token前缀，它们的token序列 `y_{i,<t}` 是完全相同的。因此，对于这些共享的token，它们的 `r_{i,t}(θ)` 值（重要性采样比率）在所有分支轨迹中也是**完全相同的**。
- 对于在分支点**之后**的、每条轨迹都**独有**的token，它们的 `y_{i,<t}` 不同，因此 `r_{i,t}(θ)` 也是**不同的**。

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{(q,a)\sim D,\{y_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|q)} \left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|y_i|}\sum_{t=1}^{|y_i|}\min\left(r_{i,t}(\theta)\hat{A}_{i,t},\right.\right. \\ \left.\left.\text{clip}\left(r_{i,t}(\theta),1-\epsilon,1+\epsilon\right)\hat{A}_{i,t}\right) - \beta D_{\text{KL}}(\pi_\theta \,\|\, \pi_{\text{ref}})\right]. \tag{6}$$

$$r_{i,t}(\theta) = \frac{\pi_\theta(y_{i,t} \mid x, y_{i,<t})}{\pi_{\text{ref}}(y_{i,t} \mid x, y_{i,<t})}, \begin{cases} r_{i,t}(\theta) = r_{j,t}(\theta), & \text{if } y_{i,<t} = y_{j,<t} \text{ (i.e., shared tokens)} \\ r_{i,t}(\theta) \neq r_{j,t}(\theta), & \text{if } y_{i,<t} \neq y_{j,<t} \text{ (i.e., individual tokens)} \end{cases}$$

Hierarchical Reward Design

使用了准确率+格式的分层奖励。另外，如果模型生成了正确答案，并且在推理过程中使用了工具，则会给予额外的奖励。

Algorithm 1 Agentic Reinforce Policy Optimization

Input initial policy model $\pi_{\theta_{\text{init}}}$; reward models r_ϕ; task prompts $\mathcal{D}$; hyperparameters ϵ, α, β, μ, τ, M, N, Z, k

policy model $\pi_\theta \leftarrow \pi_{\theta_{\text{init}}}$
for iteration = 1, . . . , I **do**
  reference model $\pi_{ref} \leftarrow \pi_\theta$
  for step = 1, . . . , S **do**
    Sample a batch $\mathcal{D}_b$ from $\mathcal{D}$
    Update the old policy model $\pi_{\theta_{old}} \leftarrow \pi_\theta$
    Sample N reasoning 1-step paths $\{y_i\}_{i=1}^{N} \sim \pi_{\theta_{old}}(\cdot \mid q)$ for each question $q \in \mathcal{D}_b$
    Compute initial entropy $H_{i,initial}$ of the first k tokens in each outputs
    let all rollouts $\{rollout_i\} \leftarrow \{y_i\}_{i=1}^{N}$
    tool-call step $t \leftarrow 1$
    while Any unfinished y_i **do**
      Parse unfinished y_i, execute tools and obtain results $d_i = T(y_i)$
      Insert d into rollout $y_i \leftarrow y_i + d_i$
      Generate k additional tokens based on inserted y_i to compute step-level entropy $H_{i,t}$
      Compute normalized change in entropy $\Delta H_{i,t} = Normalize(H_{i,t} - H_{i,inintial})$ for each y_i
      Compute partial sampling probability $P_{i,t} = \alpha + \beta \cdot \Delta H_{i,t}$
      if $P_{i,t} > \tau$ **then**
        Branch out Z additional rollouts $\{y_i\}_{i}^{Z}$ and add them to $\{rollout_i\}$
      if $|y_i| = M$ **then**
        Sample $\{y_i\}$ until y_i produce the final answer
      else
        Sample $\{y_i\}$ to produce new reasoning step
        $t \leftarrow t + 1$
    if $|y_i| < M$ **then**
      Sample $M - |y_i|$ additional independent rollouts and add them to $\{rollout_i\}$
      Compute rewards $\{r_i\}_{i=1}^{M}$ for each sampled rollouts y_i by running r_ϕ
      Compute $\hat{A}_{i,t}$ for the t-th token of o_i through group relative advantage estimation.
    for GRPO iteration = 1, . . . , μ **do**
      Update the policy model π_θ by maximizing the GRPO objective (Equation 6)
  Update r_ϕ through continuous training using a replay mechanism.

Output π_θ

实验：

作者在三大类、共13个极具挑战性的基准测试上对ARPO进行了全面评估。

- **实验设置**：
  - **任务领域**：计算推理（AIME, MATH）、知识密集型推理（HotpotQA, WebWalker）、深度搜索（GAIA, HLE, xbench）。
  - **基准模型**：涵盖了Qwen和Llama系列的不同尺寸模型。
  - **对比算法**：包括传统的轨迹级RL算法（GRPO, DAPO, Reinforce++）和基于提示的方法等。
- **训练设置**：
  - SFT：利用 Tool-Star 的 54K 训练样本的开源数据集和数学推理数据集STILL（0.8K）在LLaMAFactory上进行微调。
  - RL：

- 深度推理任务：这包括计算推理（例如，AIME24，MATH500）和基于多跳知识的推理（例如，HotpotQA，Bamboogle）。 使用 Tool-Star 的 10K 开源 RL 训练样本进行算法比较。
- 深度搜索任务：这些任务需要广泛的 Web 探索和信息整合，需要更长的上下文和频繁的工具交互。 使用来自 SimpleDeepSearcher 和 WebSailor 的 1K 混合硬搜索样本 用于训练。

1. Deep Reasoning Tasks: For models with 7B parameters, whether using ARPO or other trajectory-level RL methods, our standard setup includes a total training batch size of 128, a PPO mini-batch size of 16, a global rollout size of 16, and an initial sampling size of 8. Each interaction response length is capped at 4096 tokens. For ARPO rollouts, we set the entropy weight to 0.2, the parameter a to 0.5, and the threshold to 0.5. To stabilize training, the KL divergence coefficient in GRPO is set to 0. The reinforcement learning phase spans 2 epochs, conducted on 8 NVIDIA H800 GPUs.

2. Deep Search Tasks: For models with 8B parameters, we maintain the same settings as in the Deep Reasoning Tasks, except that each interaction response length is extended to 8192 tokens. For 14B models, the same parameters are used, but experiments are conducted on 16 NVIDIA H800 GPUs. Due to a limited dataset of 1K samples, the reinforcement learning phase lasts for 5 epochs.

- **核心实验结果**：

a. **性能全面领先**：在所有任务和模型上，ARPO的性能稳定且显著地优于所有轨迹级RL基线算法，平均准确率提升约4%。

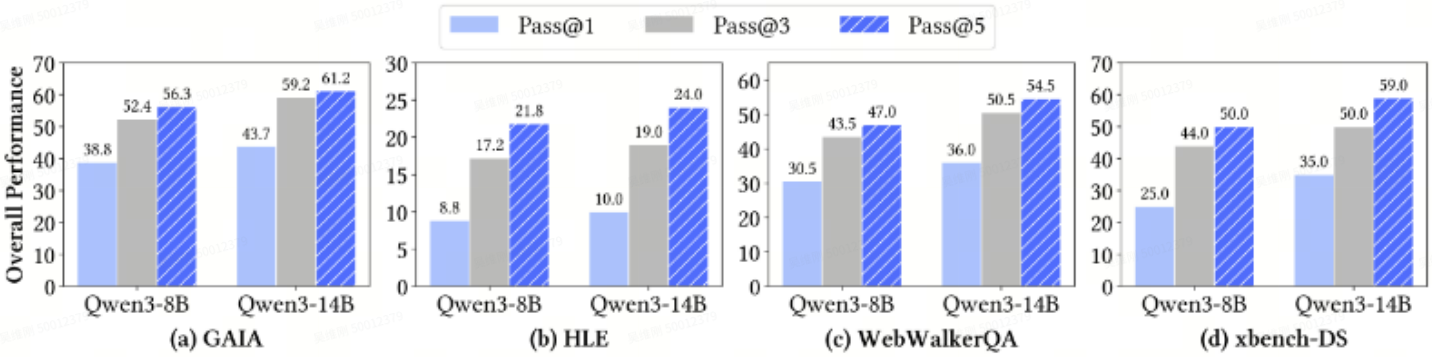
b. **惊人的工具调用效率**：ARPO最引人注目的优势之一是其**效率**。实验表明，ARPO**仅使用GRPO一半的工具调用预算**，就能达到甚至超越后者的性能（如图1右和图7所示）。这在实际应用中意味着巨大的成本节约。

c. **在深度搜索任务中表现卓越**：在需要长程规划和频繁工具交互的深度搜索任务（如GAIA）中，ARPO的优势尤为明显。这证明了其步级探索机制对于解决复杂长链条任务至关重要。

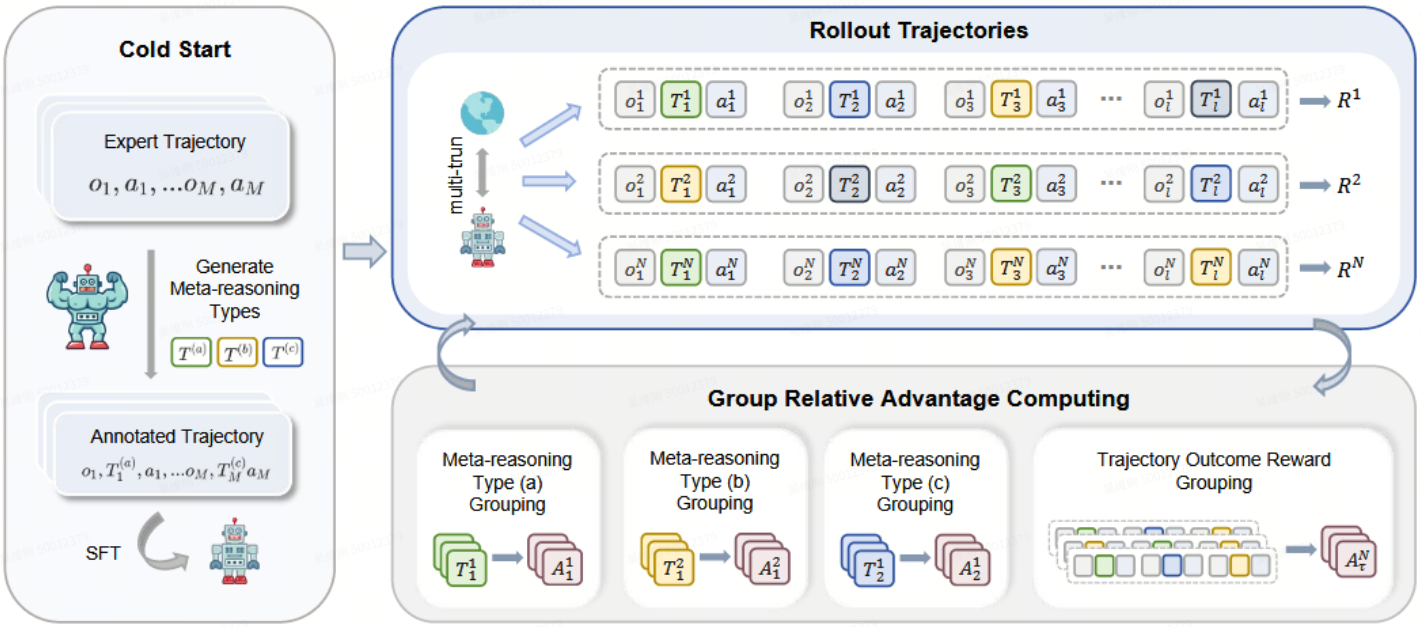
d. **优秀的扩展性 (Scaling)**：通过Pass@K（采样K次通过率）分析，发现ARPO不仅在Pass@1上表现好，在Pass@3和Pass@5上也有一致的性能提升，表明其能生成更多样化且高质量的解。

Method	Mathematical Reasoning					Knowledge-Intensive Reasoning					Avg.
	AIME24	AIME25	MATH500	GSM8K	MATH	WebWalker	HQA	2Wiki.	MuSiQ.	Bamb.	
Qwen2.5-3B-Instruct	10.0	6.7	63.0	75.0	71.6	0.5	9.7	9.4	3.6	11.7	26.1
+ TIR Prompting	6.7	6.7	52.2	56.6	62.8	14.0	15.4	14.1	6.1	16.4	25.1
+ GRPO	20.0	13.3	**72.0**	**86.0**	81.0	21.0	56.5	64.5	24.7	65.2	50.4
+ Reinforce ++	16.7	13.3	70.4	85.0	80.2	19.5	55.9	62.3	27.9	65.7	49.7
+ DAPO	20.0	16.7	71.2	85.0	81.2	19.5	54.8	62.5	**30.0**	64.8	50.6
+ ARPO	**23.3**	**20.0**	71.4	85.0	**82.5**	**24.5**	**58.5**	**67.4**	28.7	**66.8**	**52.8**
Llama3.1-8B-Instruct	3.3	0.0	43.3	81.4	60.6	3.0	24.3	24.6	10.4	40.0	28.8
+ TIR Prompting	3.3	3.3	39.4	73.8	58.2	15.0	48.5	47.5	15.5	58.4	36.3
+ GRPO	13.3	13.3	62.4	87.4	79.2	26.5	57.8	71.8	31.0	68.2	51.1
+ Reinforce ++	13.3	**16.7**	61.4	87.0	77.2	27.5	57.1	71.6	29.9	69.1	51.1
+ DAPO	16.7	13.3	61.2	87.4	76.4	25.5	56.6	70.3	29.2	67.3	50.4
+ ARPO	**23.3**	**16.7**	**64.6**	**88.0**	**80.2**	**30.5**	**65.4**	**75.5**	**34.8**	**73.8**	**55.3**
Qwen2.5-7B-Instruct	10.0	10.0	70.6	90.2	82.0	2.0	12.2	12.6	6.6	24.0	32.0
+ TIR Prompting	6.7	10.0	68.2	64.6	78.2	15.5	14.8	18.3	9.5	23.6	31.0
+ GRPO	23.3	26.7	78.0	**92.8**	87.8	22.0	**59.0**	**76.1**	30.6	68.4	56.5
+ Reinforce ++	26.7	23.3	78.0	92.2	**88.8**	**26.0**	55.1	68.9	25.2	64.9	54.9
+ DAPO	20.0	23.3	**80.4**	91.0	**88.8**	24.0	57.7	68.4	28.6	65.5	54.8
+ ARPO	**30.0**	**30.0**	78.8	92.2	**88.8**	**26.0**	58.8	**76.1**	**31.1**	**71.5**	**58.3**

Method	General AI Assistant				WebWalkerQA				Humanity's Last Exam				XBench
	Lv.1	Lv.2	Lv.3	Avg.	Easy	Med.	Hard	Avg.	NS	CE	SF	Avg.	Avg.
Direct Reasoning (>=32B)													
Qwen3-32B-thinking	26.2	12.1	0	14.9	6.9	1.1	2.9	3.1	14.6	9.8	8.4	12.6	14.0
DeepSeek-R1-32B	21.5	13.6	0.0	14.2	7.5	1.4	4.2	3.8	6.6	5.1	6.5	6.4	10.0
QwQ-32B	30.9	6.5	5.2	18.9	7.5	2.1	4.6	4.3	11.5	7.3	5.2	9.6	10.7
GPT-4o	23.1	15.4	8.3	17.5	6.7	6.0	4.2	5.5	2.7	1.2	3.2	2.6	18.0
DeepSeek-R1-671B	40.5	21.2	5.2	25.2	5.0	11.8	11.3	10.0	8.5	8.1	9.3	8.6	32.7
o1-preview†	-	-	-	-	11.9	10.4	7.9	9.9	12.9	8.1	6.6	11.1	-
Single-Enhanced Method (Qwen3-8B)													
Vanilla RAG	28.2	15.4	**16.7**	20.4	8.9	10.7	9.9	10.0	5.1	1.6	12.9	5.8	8.0
Search-o1	35.9	15.4	0.0	21.4	6.7	15.5	9.7	11.5	7.6	2.7	5.3	6.4	10.0
WebThinker	43.6	11.5	0.0	22.3	6.7	13.1	16.9	13.0	7.3	4.0	6.3	6.6	13.0
ReAct	35.9	17.3	8.3	23.3	8.9	16.7	18.3	15.5	4.2	4.0	6.3	4.6	16.0
RL-based Method (Qwen3-8B)													
Qwen3-8B	28.1	15.4	**16.7**	20.4	0.0	2.4	2.8	2.0	3.9	2.7	8.4	4.6	9.0
+ GRPO	48.7	25.0	8.3	32.0	24.4	**33.3**	26.8	29.0	**7.9**	4.0	10.5	7.8	20.0
+ ARPO	**53.9**	**32.7**	**16.7**	**38.8**	**26.7**	**33.3**	**29.6**	**30.5**	7.3	**6.7**	**15.8**	**8.8**	**25.0**
Single-Enhanced Method (Qwen3-14B)													
Vanilla RAG	38.5	19.2	8.3	25.2	17.8	13.1	11.3	13.5	5.5	6.3	9.4	6.0	15.0
Search-o1	48.7	23.1	0.0	30.1	11.1	21.4	16.9	17.5	6.4	4.0	10.5	6.8	21.0
WebThinker	48.7	26.9	8.3	33.0	13.3	23.8	18.3	19.5	7.0	4.0	9.5	7.0	23.0
ReAct	48.7	25.0	8.3	32.0	11.1	20.2	12.7	15.5	5.8	5.3	10.5	6.6	20.0
RL-based Method (Qwen3-14B)													
Qwen3-14B	33.3	13.5	0.0	19.4	6.7	2.4	4.2	4.0	5.5	6.7	11.6	6.8	14.0
+ GRPO	51.3	34.6	0.0	36.9	28.9	33.3	26.8	30.0	7.9	6.7	12.6	8.6	27.0
+ ARPO	**56.4**	**40.4**	**16.7**	**43.7**	**31.1**	**42.9**	**31.0**	**36.0**	**10.3**	**10.7**	**13.7**	**10.0**	**32.0**



RLVMR: Reinforcement Learning with Verifiable Meta-Reasoning Rewards for Robust Long-Horizon Agents



动机：

现有的RL方法通常只优化最终任务的成功率，而忽略了对底层推理过程的监督。这导致了以下问题：

- **低效探索**：即使代理（agent）最终达到了目标，其推理路径可能也是有缺陷的、低效的，甚至包含逻辑错误。这种情况下，代理在面对未见过的任务时往往无法泛化，因为它们没有学会如何进行连贯的推理。
- **策略脆弱性**：由于缺乏对推理过程的监督，代理在训练过程中可能会学到一些脆弱的策略，这些策略在新环境中容易失败。

提出了RLVMR的框架，通过在端到端的RL中整合密集的过程级监督，直接奖励可验证的元推理（meta-reasoning）行为，从而提升Agent的推理质量，使其更加健壮、高效和可解释。

核心思想

让agent学习明确地表达其“元认知”（meta-cognition）状态，如规划、探索、反思等，对这些思考过程给予可验证的奖励，让模型学到更高效的推理策略。

方法

他们将原本的单一的思考过程，细化为四种不同的元认知标签。

- <planning>：用于任务开始或需要重新规划时。
- <explore>：遇到不确定性或瓶颈时，探索新的可能性。
- <reflection>：之前的动作无效时，分析错误原因，并给出修正方案。
- <monitor>：常规执行过程中，跟踪当前子目标和任务进展。

1. 冷启动SFT

让模型学会理解和使用上述标签。

首先，收集一批专家轨迹（只有观察和动作，<o1, a1, ...>），让gpt-4对这些轨迹的每一步都打上一个合适的标签，生成包含推理过程的数据集（包含观察、思考和动作，<o1,T1,a1, ...>），用来微调agent

2. RLVMR

- Meta-Reasoning-Aware Reward Shaping

结果奖励：二元奖励，任务成功与否

元推理奖励 r_t^{MR} ：每一步一个奖励

$r_{planning}$: 如果一个planning步骤最终导向成功，给予奖励

$r_{explore}$: 当前动作探索了新的对象或位置

$r_{relection}$: 在一个失败序列后，引导了一个修正动作

格式惩罚 r_t^{format} ： 不满足<tag>...<action>...</action>

- Group Relative Policy Optimization with Meta-Reasoning (GRPO-MR)

轨迹级的相对优势

$$A_k^{\text{traj}} = \frac{R(\tau_k) - \mu_R}{\sigma_R},$$

meta-reasoning级的相对优势

$$A_{t,\text{tag}}^{\text{MR}} = \frac{r_{t,\text{tag}}^{\text{MR}} - \mu_{\text{tag}}}{\sigma_{\text{tag}}},$$

最终step-level的优势为

$$A_t = \alpha \cdot A_k^{\text{traj}} + (1-\alpha) \cdot A_{t,\text{tag}}^{\text{MR}},$$

$$\mathcal{L}_{\text{final}} = \mathbb{E}_t \left[\min \left(r_t(\theta) A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t \right) \right] - \lambda_{\text{KL}} D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}),$$

结果

在两个具有挑战性的长视野基准测试（ALFWorld和ScienceWorld）上验证了RLVMR的有效性。实验结果表明，RLVMR在所有设置中均取得了新的最佳性能，特别是在最困难的未见任务分割（L2）上，7B模型的成功率达到了83.6%。此外，RLVMR训练的代理在推理质量上也有显著提升，表现为显著减少了重复和无效动作，增强了错误恢复能力。

- **L0（seen-L0）**：已见任务变体和类别。
- **L1（unseen-L1）**：未见任务变体，但已见任务类别。
- **L2（unseen-L2）**：未见任务变体和类别。

Model	Method	ALFWorld			ScienceWorld		
		L0	L1	L2	L0	L1	L2
AgentGym	SFT+RL	76.6	63.3	–	46.9	33.6	–
GPT-4o	ReAct	57.3	66.0	68.8	45.4	49.2	41.0
DeepSeek-V3	ReAct	60.2	65.9	53.9	27.3	35.2	26.5
DeepSeek-R1	ReAct	68.8	70.2	67.3	22.2	31.4	29.1
Qwen-1.5B	ReAct	11.3	13.7	10.2	1.2	0.8	0.8
	+ SFT	43.0	38.7	17.6	20.3	18.0	12.5
	+ ETO	64.1	66.4	25.8	39.1	22.7	15.6
	+ GRPO	76.6	71.1	29.7	21.1	13.7	10.9
	+ GiGPO	86.7	83.2	48.0	25.8	15.2	4.7
	+ RLVMR	**89.1**	**87.9**	**56.3**	**46.9**	**34.4**	**26.5**
Qwen-7B	ReAct	23.1	28.5	27.0	7.8	11.3	6.3
	+ SFT	63.3	57.0	37.5	36.7	32.0	23.4
	+ ETO	70.3	74.2	51.6	62.5	40.6	28.1
	+ GRPO	79.3	77.3	52.3	49.1	30.1	26.6
	+ GiGPO	89.5	90.2	67.2	53.4	35.2	25.8
	+ RLVMR	**91.4**	**91.8**	**83.6**	**67.2**	**43.0**	**32.2**